Northeastern University - Seattle

Khoury College of
Computer Sciences

CS6200 Information Retrieval Fall 2019

Lecture 3:
Acquiring Data

Sep 16, 2019

ADMINISTRIVIA

Classroom, class time

- You are here, so you know:
  - Classroom stays the same, at 307/225 Terry Ave N
  - Class timings stay the same, 6:15pm to 9:15pm Mondays

Survey Analysis

30 responses, thanks

1. Favorite search engine (SE):
   - Not surprisingly Google: 22; Bing, Baidu, DuckDuckGo: 2 each, surprisingly ElasticSearch and Google Scholar 1 each
     [I meant web search engines, though I did not state that!]
   - Surprising reasons (apart from fast, accurate etc.):
     - "uses most advanced techniques", "most accurate SE so far", "only option in the US" -- ???
2. Goal: learn more, implement more, also "use Python to do fun stuff" !!
3. Most have used Python (except for one user, who needs to learn Python soon). A few have used Lucene/Solr, ElasticSearch.

Response to Survey Results

Great, love your responses and goals.
"Search experience" -- depth of emotion there!

We will work on ensuring you understand the technology behind SEs and have the tools to evaluate them

Hopefully, we will do fun stuff with Python!

Challenge: try a different SE along with your favorite SE for at least one query a day (during this course) and compare it to results from your favorite SE. Think about the differences in results between engines.

Quiz 1 – typical results

- Q1. What is your favorite feature in a search engine? Describe it in a sentence.
  Possible answer: Boolean Querying. Ability to use '+' and '–' along with words in the query.
  E.g. [Julius Caesar +Shakespeare –quotes] should return the Shakespeare's Julius Caesar, without any quote sites. Note the use of […] to indicate a query.
  https://www.bing.com/search?q=Julius+Caesar+%2BShakespeare+-quotes
  vs https://www.bing.com/search?q=Julius+Caesar+%2BShakespeare

- Q2. Name one component of a search engine that you do not normally see in a database system. In one or two sentences, describe what it does.
  Possible answer: Unstructured Querying. Ability to take queries close to human language.

- Q3. In the context of IR, what does the acronym SERP stands for? …
  Answer: Search Engine Result Page …

Quiz 1 …2

Q4. Comparing search engines:

a. Information need:

1. Forecast for hurricane Humberto
2. Events happening in Seattle in the coming weekend.

b. What information I expect to get to satisfy my information need:

1. Updates on a map regarding the direction of the storm and its intensity.
2. All the events happening in Seattle for the upcoming weekend.

c. Query corresponding to the information need:

Hurricane Humberto Forecast

Seattle weekend events

Quiz 1 …3

d. Got information that I wanted to get:

Yes.

e. Query1: Comments on results got from Bing and Google for this query:

Bing: Retrieved articles about the storm with maps. What I wanted.

Google: Retrieved only articles regarding the storm but no maps. Good Job.

Query2: Comments on results got from Bing and Google for this query:

Query2:

Both search Engines did a good job with providing URLs like “thestranger.com” where I can look up the relevant events.

Ordering of the results were quite different for both.

Quiz 1 …4

Q5. Comparing search engines:

a. Criteria:

Possible answer: Resolving the intent of the query and its context.

Possible answer: The ranking of the results. The order of relevant results may misguide the user with irrelevant information.

b. Was the evaluation easy? If not, what makes it complicated?

Possible answer: No. *Human subjectivity on what is relevant and the context of time*.

Your quiz answers + my comments … 1

Some confusion about fave feature. People have included indexer, ability to search on anything… I was expected more like site-specific search, search by image etc.

Most say evaluation is not easy

Several have done only 1 info need, not 2. Please read the questions.

Your quiz answers + my comments … 2

Some interesting queries:

[How can I find a good person to marry]

[Grocery stores near Green Lake]

[President], [James], [Buffalo]

For many queries, responses say Bing and Google gave similar results. More people report Bing gave better results than the number who indicated that Bing was their fave. **What we learn:** we need to compare, and have tools to measure goodness

Evaluation is hard, yes!

Questions?

Overview

Web Crawling

How Crawlers work
Freshness and Age
Crawls vs. feeds

Other Crawlers

Document Conversion

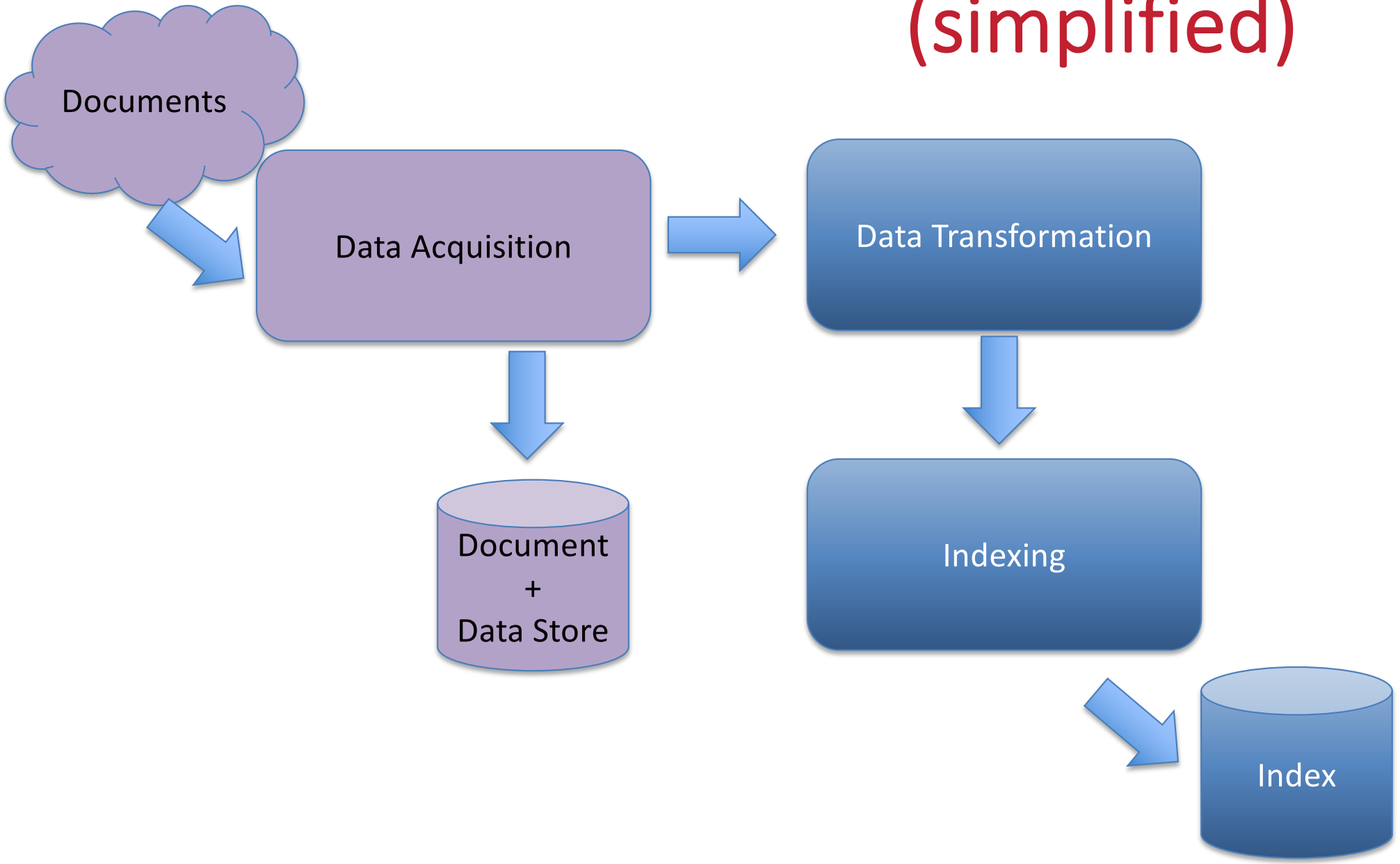
Document Storage

Dealing with Duplicates

Removing Noise

CRAWLING THE WEB

Search Engine: Behind the Scenes (simplified)



Crawling the Web

Issues

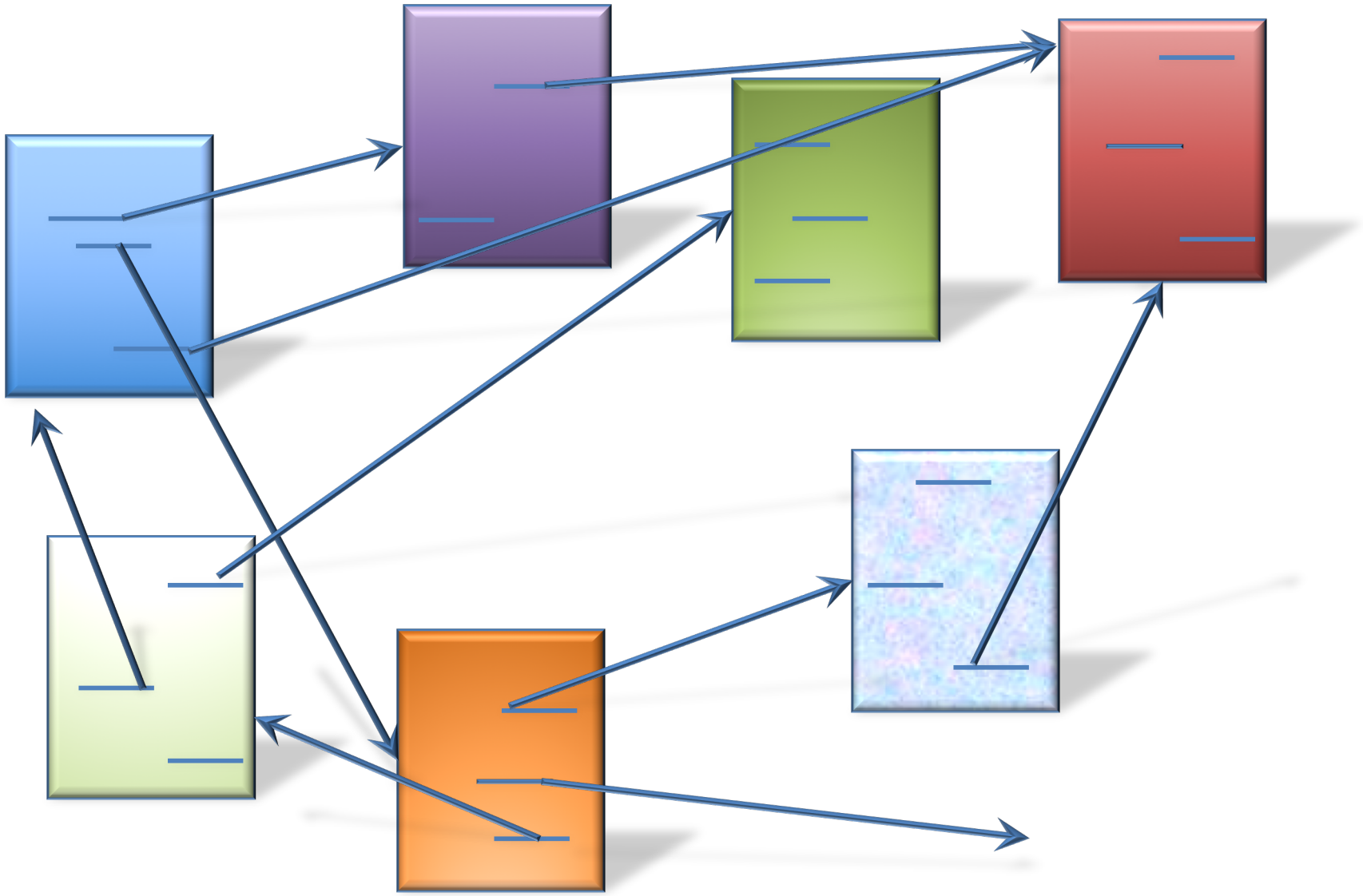
- **Size**: The Web is HUGE and growing all the time
- **Access**: Search engines do not control web pages
- **Change**: Pages are constantly changing
- **Data**: Pages are of different, often mixed, types

But:

- Pages often linked (in very complicated ways), and that's informative!
- A **lot** of pages open to all, easy to access

HOW DO WE GET PAGES FROM A WEB SITE?

A subset of the Web

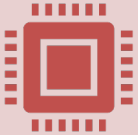


URLs and HTTP

URL: Every web page has a unique **Uniform Resource Locator**

Specifies protocol, hostname and file name

E.g. https://piazza.com/northeastern/fall2019/cs6200seattle/home



Web pages stored on (web) servers;
HTTP: Hypertext Transfer Protocol used to exchange information between server and (web) client software

Client (e.g. crawler) submits a HTTP **request** message to server, and gets back a **response**

Requesting a web page

Web crawler client initiates request by establishing a TCP connection

Typically to port 80 on server
Typically using a GET method

Server listening on that port, scans request parameters

Server returns status and a response message

Body of response typically the requested page
[or error/info message]

Note: Some pages may require redirects to be followed

Sample GET request and response

For URL: http://www.example.com/index.html

Request

GET **/index.html** **HTTP**/1.1
Host: www.example.com

Response

HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

Actual content of page

<**html**> <**head**> <**title**>An Example Page</**title**> </**head**> <**body**> Hello World, this is a very simple HTML document. </**body**> </**html**>

Adapted from: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Web Crawling Algorithm

1. Start with a set of *seed URLs* (algorithm parameter)
   a. Where do we get them from?
2. Add seed URLs to URL queue ('frontier')
3. Fetch pages from queue
   a. usually many pages fetched in parallel
4. Download and parse pages, process 'link tags' to extract other URLS to fetch
5. Add new URLs to frontier
6. Continue crawling till no more URLs in frontier (or if enough URLs crawled, or until disk full)

Crawling in Parallel …

Takes time for responses to be returned for requests

Crawlers have to wait a lot

So: web crawlers use a large number of threads and fetch hundreds of pages in parallel

But: …

Politeness Policies

Crawler's requests for hundreds of pages from the same domain could flood (small) sites

The domain pages cannot do their main job

To avoid this, crawlers use politeness policies. Users control themselves to avoid getting prohibited

Add a delay between requests to the same server

Fetch breadth-wise instead of depth-wise

Sites may request crawlers to police themselves using robots.txt

Robots.txt

- Web site admins may want to control access to (parts of) their data
- Can create a specific **robots.txt** file at site root:

```
User-agent: *
Disallow: /confidential/
Disallow: /personal/
Allow: /public/

User-agent: MyFavoriteBot
Disallow:

Sitemap: http://mysite.com/sitemap.xml.gz
```

By default, all crawlers allowed to folder /public, but not to /confidential and /personal

Crawler named MyFavoriteBot allowed access to every folder

More about sitemaps soon

- Crawlers expected to adhere to robots.txt

Crawler Thread Pseudo-code

```
procedure CrawlerThread(frontier)
        while NOT frontier.empty() do
                website ← frontier.nextSite()
                url ← website.nextUrl()

                if website.permitsCrawl(url) then
                        text <- getUrlContent(url)
                        storeDocument(url, text, time…)
                        for each tUrl in parse(text) do
                                frontier.addUrl(tUrl)
                        end for
                end if

                frontier.removeSite(website)
        end while
end procedure
```

Notes on crawling

- Where do we get seed URLs from?
  - Can start with one URL or a few and build up a bigger seed list
  - Could determine nature of collection
    - What if we started from **wikipedia.org** ?
    - Or what if we started from **espn.com** ?

- Not enough to crawl pages once, but how often should we crawl pages?

  → Discussion on Freshness and Age

FRESHNESS AND AGE

Freshness of pages

Web pages constantly changing: pages added, deleted, modified

A page is **fresh** if the crawl has most recent version, otherwise it is **stale**

Web crawler must revisit pages continually

To see if they have changed (i.e. if they have become **Stale**)

And if so, to re-crawl them, to keep them **Fresh**

Freshness: the HEAD request

- Special request type in HTTP named HEAD
  - Makes it easy to check for page changes
  - Returns page 'metadata' – *data about the page, not the page contents*

Client request:

```
HEAD / HTTP/1.1
Host: http://www.northeastern.edu
```

Server response:

```
HTTP/1.1 200 OK
X-Powered-By: PHP/7.2.8
Access-Control-Allow-Origin: *
Content-Type: text/html; charset=UTF-8
Cache-Control: max-age=0
Expires: Sun, 15 Sep 2019 05:29:42 GMT
Date: Sun, 15 Sep 2019 05:29:42 GMT
Connection: keep-alive
```

Freshness and Age

Not feasible to constantly check all pages

- Instead, check "HOT" pages:
  important and frequently-changing pages
- Example: whitehouse.gov, news sites, major companies

Consider two different metrics:

- **Freshness**: proportion of pages that are fresh
- **Age**: Days since a page was crawled
  If page was crawled on day x, and today is day t,
  Age = (t –x)

Freshness as a metric

- **Freshness**:
  Suppose a page is refreshed rapidly,
  e.g. every minute
  - A crawled copy is almost always stale
  - No incentive for an optimized crawl to crawl it, since it will almost always be stale
  - But users want pages to be crawled !

Age as a metric

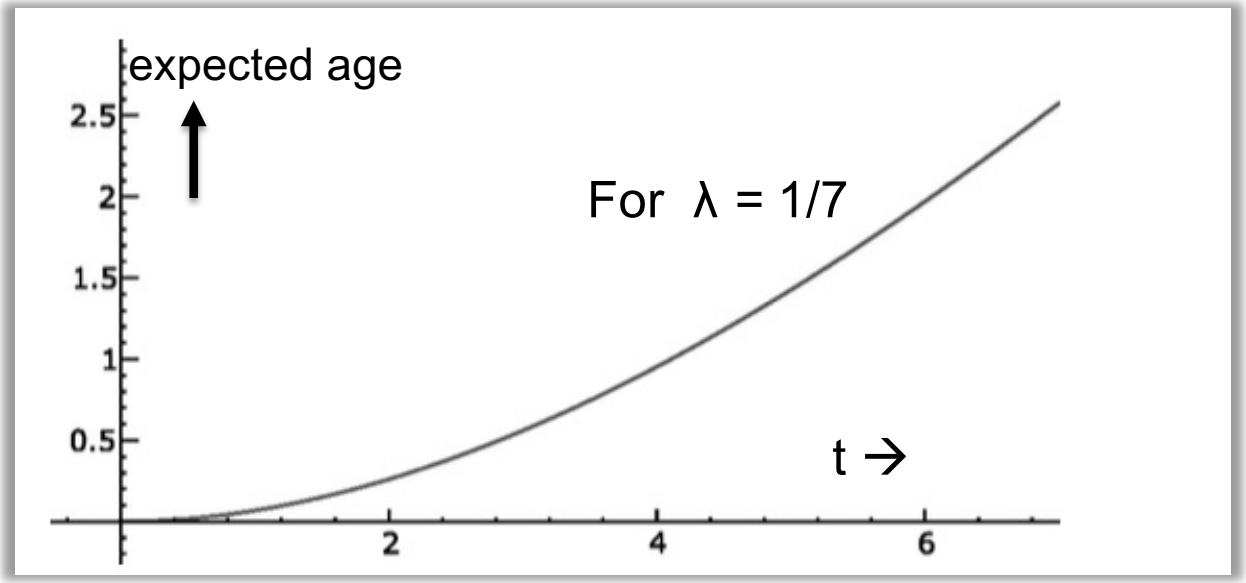
- If page changes λ times a day, expected age of page t days after it was last crawled:

$$\text{Age}(\lambda, t) = \int_0^t P(\text{page changed at time } x)(t-x)dx$$

- Web page updates follow Poisson distribution, on average [Cho & Molina, 2003].
  Plug that in to get:

Expected age of page = $\text{Age}(\lambda, t) = \int_0^t \lambda e^{-\lambda x}(t-x)dx$

Plot of Expected age of pages vs. days after crawl



- Assume pages change roughly once a week, λ = 1/7
- One week after crawl, for t = 7, expected age = ~ 2.6

- Curve goes up very fast
  Older a page gets, more it costs **not** to crawl it

MORE ON CRAWLING

Focused or Topical Crawling ..1

Focused crawling: Only crawling pages about a specific topic

Why do we need focused crawling?

Focused or Topical Crawling ..2

Goal: crawling only pages **about** a particular **topic**

Used in vertical or topical search applications

E.g. photography search, real estate search, academic search …

Pages about a topic tend to have links to pages about the same topic

So start with popular pages for a topic as seeds

Can use a text classifier to decide if a page is about a specific topic

Deep Web … 1

Not all sites/pages on the Web are crawlable

Deep (or Hidden) Web:

- Sites difficult for crawler to find
- Much larger than (accessible) Web

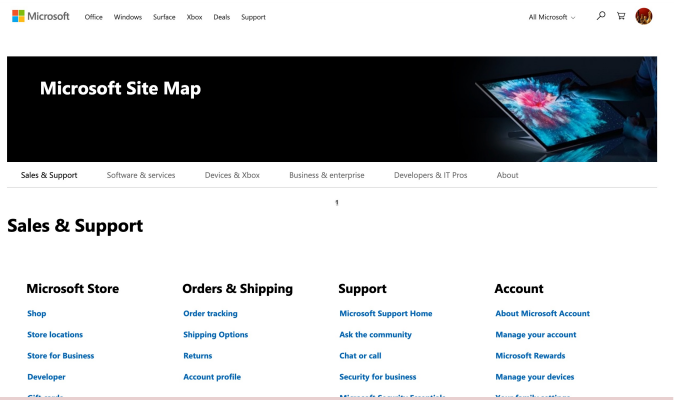
Different from Dark Web

Deep Web … 2

Three broad categories of Deep Web sites:

- Private sites: may be behind paywall, need an account or subscription
  - E.g. News sites with subscription, ACM Digital Library
- Form results: sites reachable only after data entered into a form
  - E.g. airline reservation sites; hard to crawl, too many options
- Scripted pages: pages using JavaScript, Flash etc. to generate links

Sitemaps



Different from Site Maps for people
e.g. https://www.microsoft.com/en-us/sitemap.aspx

Sitemaps meant for crawlers, used by sites:

To indicate which files to crawl (and some may be hard to find)

When they were last changed

With what frequency they change

Sitemaps can be listed in robots.txt

Sitemap Example

```
<?xml version="1.0" encoding="utf-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.sitemaps.org/schemas/
          sitemap/0.9 http://www.sitemaps.org/schemas/
          sitemap/ 0.9/sitemap.xsd">
         <url>
            <loc>http://example.com/</loc>
            <lastmod>2016-11-18</lastmod>
            <changefreq>daily</changefreq>
            <priority>0.8</priority>
         </url>
</urlset>
```

Adapted from: https://en.wikipedia.org/wiki/Sitemaps

Distributed Crawling

Different reasons:

- Gets crawler closer to the sites it crawls
- Reduces sizes of to-be-crawled and crawled lists
- Distributes computing load
- Many queues, so faster

URLs distributed between crawlers

- Typically using hash function on host name, not on whole URL

  [Can you guess why?]

Desktop or Enterprise Crawls

- For desktop or enterprise search
- Differences from web crawling:
  - Data is local or close-by
  - Important to be up-to-date and respond quicker to updates
  - Cannot be CPU or disk ‘hogs’
  - Privacy very different; role-based access may be implemented
    - Access may be based on user’s file/folder permissions

DOCUMENT FEEDS & RSS

Document Feeds … 1

Many documents ‘published’

Created at a fixed time
and rarely updated

E.g. news articles, blog posts, etc.

Published docs from a
single source can be sent
in a time-ordered sequence:
document feed

New docs easily found
at the end of the feed

Can easily be sent to crawlers

Document Feeds … 2

- Two kinds:
  - Push feeds: Alerts subscribed users to availability of new docs
    - Common in paid news agency feeds
  - Pull feeds: Subscriber must keep ‘polling’ or checking for new docs
    - Blogs, news, search results etc.

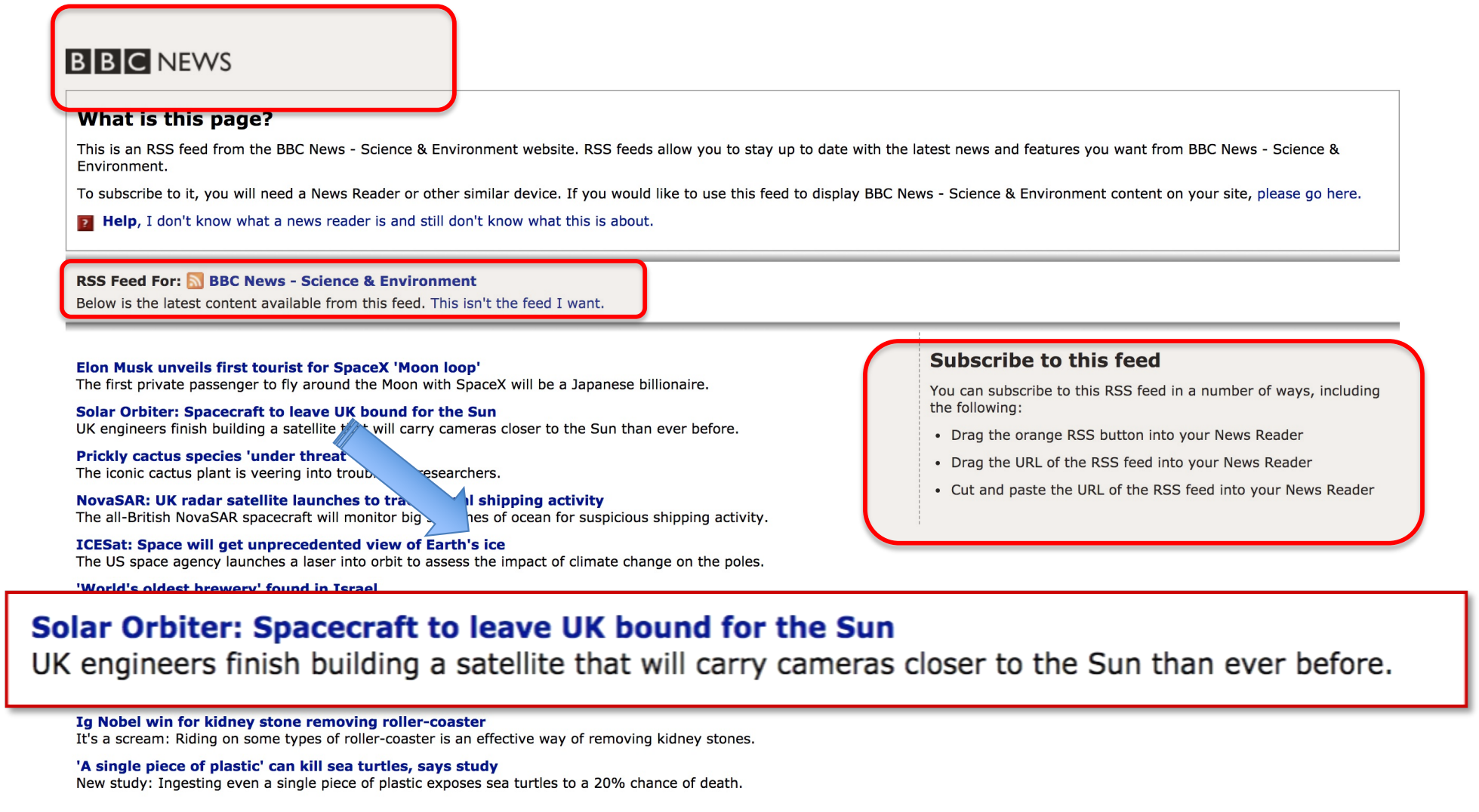
RSS: Really Simple Syndication…

- Most common pull format: RSS
  - Really Simple Syndication or RDF Site Summary or Rich Site Summary
- Competing format:

  Atom Syndication Format
- Accessed like web pages
  - using HTTP GET
  - Read using ‘newsreaders’
- Simple, easy to parse
- Easy to identify new information
- Not too popular now

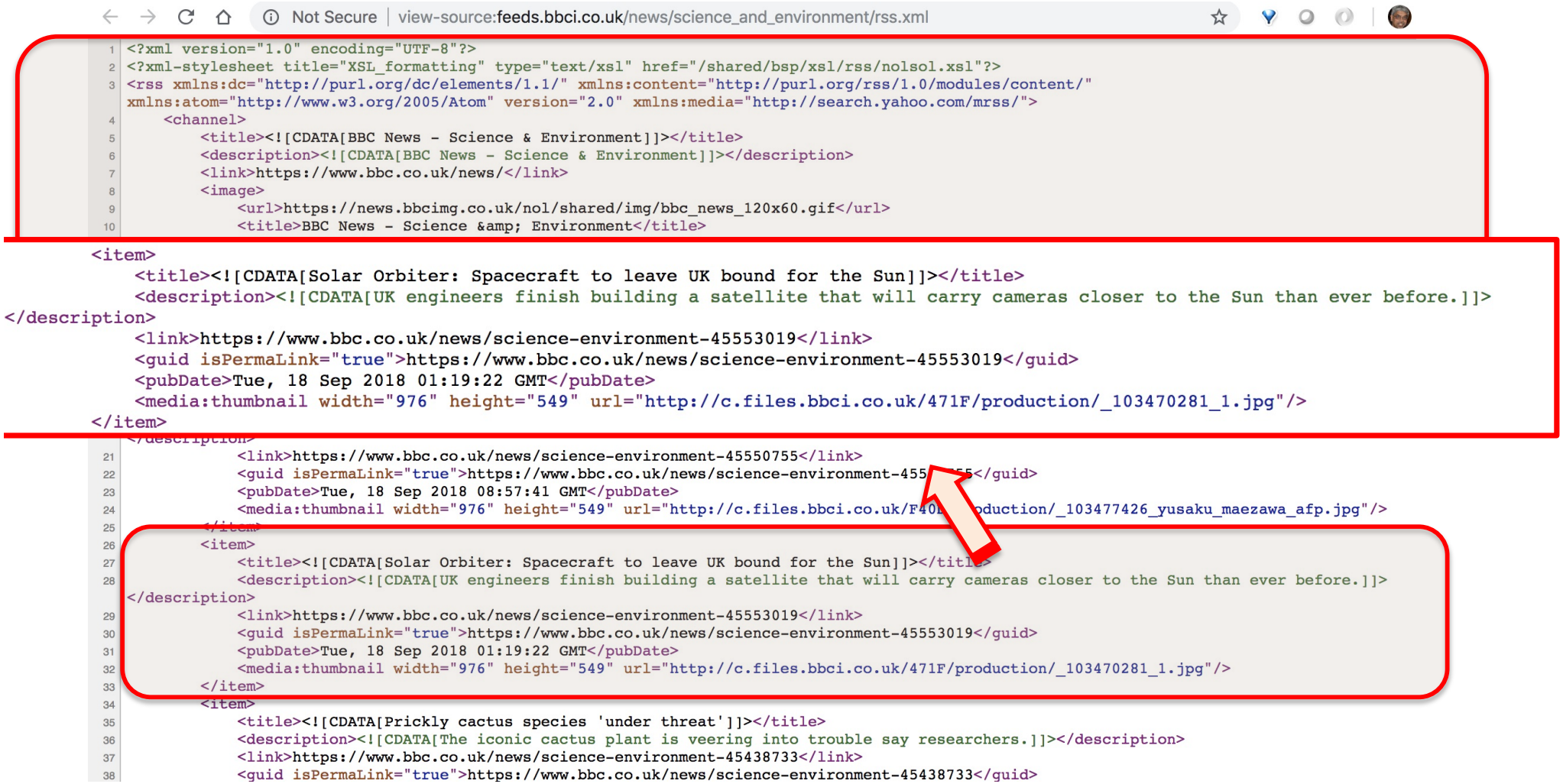
RSS Logo

RSS Example: Newsreader output



http://feeds.bbci.co.uk/news/o /rss.xml

RSS Example: Source



view-source:feeds.bbci.co.uk/news/science_and_environment/rss.xml

RSS Tags

- **Channel**
  - title
  - link
  - description
  - language
  - pubDate: Date of publication
  - ttl: Time To Live (minutes to cache data)
  - …
- **Item**
  - title
  - link
  - description
  - pubDate
  - guid
    - isPermaLink
  - media-thumbnail

Advantages of Feeds

- Good for both data owner and search engine:
  - Data owner gets data to the search engine as and when it is updated, and sends out all the data it wants to be seen (for freshness and coverage)
  - Search engine does not have to crawl repeatedly; avoids missing data or crawling data long after update

DATA CONVERSION / DATA NORMALIZATION

Data Conversion … 1

- Data comes in different formats
  - Text: raw text, PDF, HTML, Microsoft Office formats, ODF, XML, JSON,…
  - Images: JPEG, TIFF, PNG, …
  - Audio: WAV, AIFF, PCM, MP3, WMA, AAC, …
  - Video: GIF, AVI, WMV, MPEG-4, 3GPP, …
  - …

[our focus here is on text]

Data Conversion … 2

Good to use one or more conversions tools

- Convert all data to a standard, marked-up format
  E.g. HTML/XML/JSON

Goals:

- retain formatting of original
- make it possible to index and retrieve the text

Character Encodings .. 1

Text itself may be in one of many character encodings

Character Encoding:

- mapping between bit representation and glyphs (form on page)
- E.g. English: we can use ASCII ASCII 65 = ‘A’
  - 128 characters (a-z, A-Z, 0-9, special characters, control chars) stored in 7 bits,
    with extended ASCII using 8 bits providing diacritics etc.

Character Encodings .. 2

Other languages have many more glyphs

- e.g. Chinese > 40,000 chars, > 3000 in common use
- Indic languages (Devanagari script), Arabic etc.

Many languages have multiple encoding variants

- Chinese-Japanese-Korean (CJK), Indic languages, Arabic etc.
- not able to have multiple languages in a file

The Solution: Unicode

Unicode

- Single mapping from numbers to glyphs
- Attempts to include all glyphs in common use in all known languages
- But Unicode not a mapping from bits to glyphs
  - many ways to go from Unicode numbers to glyphs
  - Hence:UTF-8, UTF-16, UTF-32

Devanagari

CJK

Devanagari: http://www.unicode.org/charts/PDF/U0900.pdf
CJK: http://www.unicode.org/charts/PDF/U4E00.pdf

Unicode Transformation Format

UTF-8

- Uses 1 byte for first 128 characters
  - Identical to ASCII, compatible
- Then up to 4 bytes for other characters
- Leads to compact notation

But

- Variable lengths make string operations difficult
- Hard to jump to n^{th} character in a text

UTF-32

- Uses 4 bytes for every character
- More space required than UTF-8
- Easy to jump to any character

Space vs. ease of use

Many applications use

- UTF-32 for internal text encoding (fast random lookup)
- UTF-8 for disk storage (less space)

Unicode UTF-8 Example: Encoding π

Decimal	Hexadecimal	Encoding			
0–127	0–7F	0xxxxxxx			
128–2047	80–7FF	110xxxxx	10xxxxxx		
2048–55295	800–D7FF	1110xxxx	10xxxxxx	10xxxxxx	
55296–57343	D800–DFFF	Undefined			
57344–65535	E000–FFFF	1110xxxx	10xxxxxx	10xxxxxx	
65536–1114111	10000–10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Greek letter pi (π)

- Unicode symbol number 960
- In binary, 00000011 11000000 (3C0 in hexadecimal)
  Look at 11 bits, separate 5 higher-order bits and 6 lower-order bits
- Final encoding is **110**01111 **10**000000 (CF80 in hex)

DOCUMENT STORAGE

Why Store Documents?

- Documents available from URL, so why store it?
  - usually not stored in desktop crawls
- Many reasons to store web-crawled documents
  - saves crawling time when page is not updated
  - easy to get to text to generate snippets and extract entities and links
  - cached copies useful
- Database systems?
  - not optimized for documents of varying lengths, with different retrieval requirements
  - web search engines use customized document storage systems

Requirements for Document Storage

- Random access
  - request the content of a document based on its URL
  - hash function based on URL is typical
    - points to server and file location
- Efficient use of storage
  - large files + compression
- Ease of Updates
  - must allow for easy updates of stored documents with new crawl data
  - must allow for adding new anchor text segments etc.

Large Files

- File per document is wasteful
- Instead:
  - Use several large files
  - Store many documents in each large file
    - avoids overhead in opening and closing files
    - reduces seek time relative to read time (block reads)
- Compound documents formats
  - used to store multiple documents in a file
  - e.g., TREC Web

TREC Web Format

```
<DOC>
<DOCNO>WTX001-B01-10</DOCNO>
<DOCHDR>
http://www.example.com/test.html 204.244.59.33 19970101013145 text/html 440
HTTP/1.0 200 OK
Date: Wed, 01 Jan 1997 01:21:13 GMT
Server: Apache/1.0.3
Content-type: text/html
Content-length: 270
Last-modified: Mon, 25 Nov 1996 05:31:24 GMT
</DOCHDR>
<HTML>
<TITLE>Tropical Fish Store</TITLE>
Coming soon!
</HTML>
</DOC>
<DOC>
<DOCNO>WTX001-B01-109</DOCNO>
<DOCHDR>
http://www.example.com/fish.html 204.244.59.33 19970101013149 text/html 440
HTTP/1.0 200 OK
Date: Wed, 01 Jan 1997 01:21:19 GMT
Server: Apache/1.0.3
Content-type: text/html
Content-length: 270
Last-modified: Mon, 25 Nov 1996 05:31:24 GMT
</DOCHDR>
<HTML>
<TITLE>Fish Information</TITLE>
This page will soon contain interesting
information about tropical fish.
</HTML>
</DOC>
```

Compression … 1

- Text is highly redundant (predictable)
- Compression techniques exploit redundancy to make files smaller without losing content
- Popular algorithms can compress HTML and XML text by about 80%
  - zip, gzip, LZW

Compression … 2

Compression better with large blocks

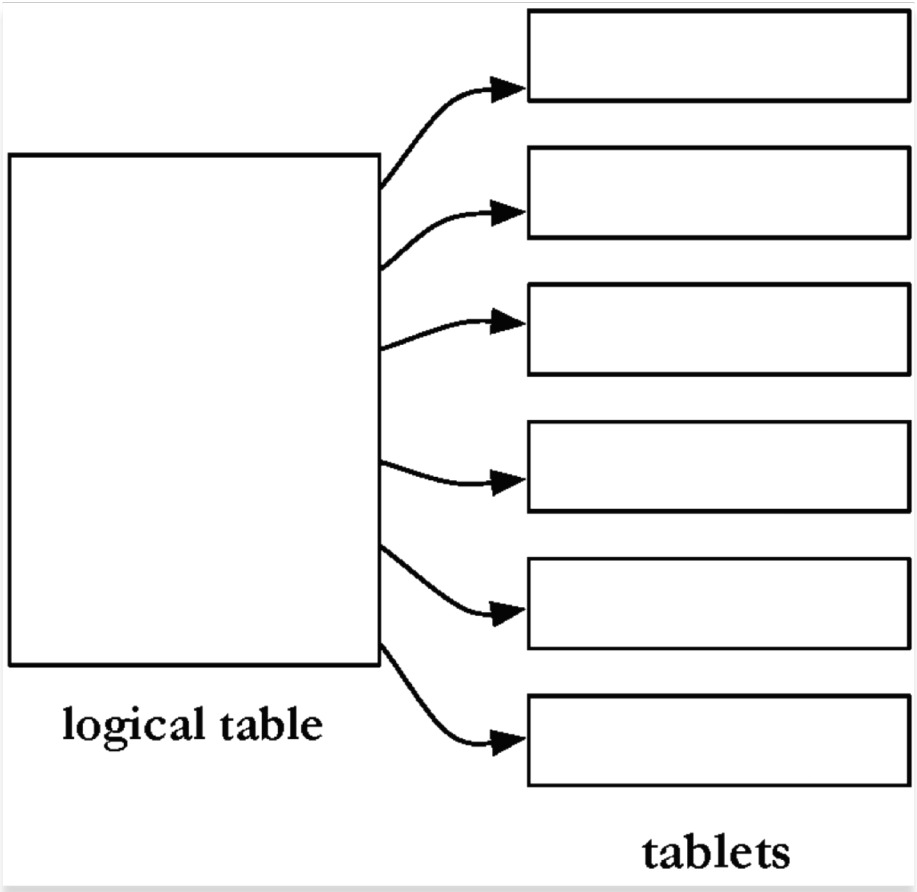
but random access is affected

Compression with smaller blocks better for speed of access (latency)

not as good for compression efficiency

BigTable ..1

- Google's document storage system
  - Customized for storing, retrieving, and updating web pages
  - Handles large collections using inexpensive computers
  - Big table split into 'tablets'

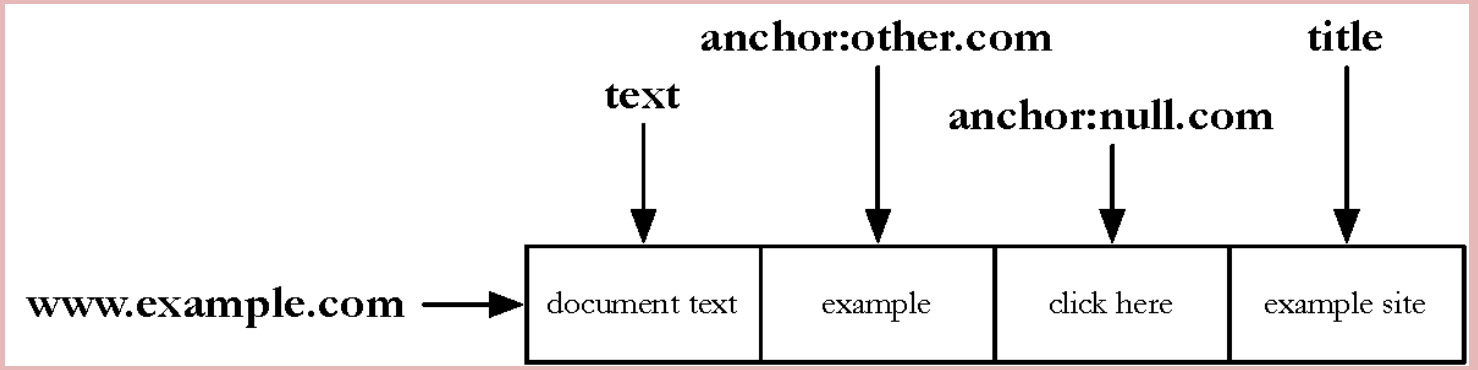


BigTable ..2

- No query language, no complex queries to optimize
- Only row-level transactions
- Data stored in files not modified
- New data stored in RAM, older data in files
- Emphasis on failure recovery
  - Tablets stored in a replicated file system accessible by all BigTable servers
  - Any changes to a BigTable tablet recorded to a transaction log, which is also stored in a shared file system
  - If any tablet server crashes, another server can immediately read the tablet data and transaction log from the file system and take over

BigTable ..3

- Logically organized into rows
- A row stores data for a single web page



- Combination of a row key, a column key, and a timestamp point to a single *cell* in the row

BigTable ..4

BigTable can have a huge number of columns per row

- all rows have the same column groups
- but not all rows have the same number of columns
- important for reducing disk reads to access document data

Rows partitioned into tablets based on their row keys

- simplifies determining which server is appropriate

Overall: emphasis on speed & scale

DETECTING DUPLICATES

Detecting Duplicates

- Duplicate and near-duplicate documents
  - Actual copies, different versions of documents, plagiarism, spam, mirror sites
- Duplicates are of little value to users
- But consume significant resources (computing, disk, memory…)
  - 30% of web pages in a large crawl are exact or near duplicates
  - Wasteful during crawling, indexing, and search

Detecting Exact Duplicates

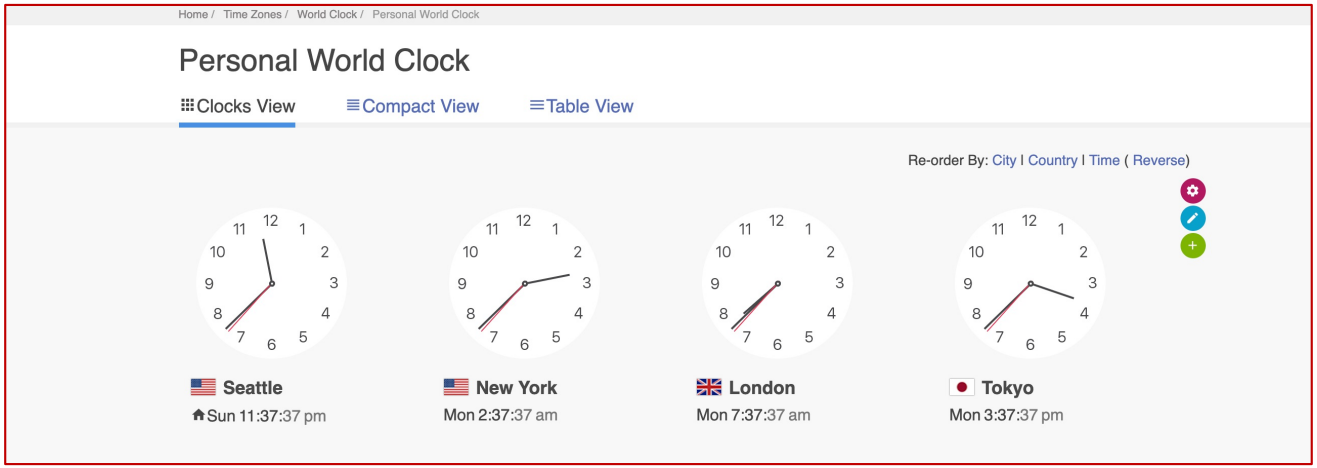
- *Exact* duplicate detection is relatively easy
- *Checksum* techniques
  - Checksum: value computed on the content of the document
    - e.g., sum of the bytes in the document file

T	r	o	p	i	c	a	l		f	i	s	h	*Sum*
54	72	6F	70	69	63	61	6C	20	66	69	73	68	508

  - But files with different text may have same checksum
- Functions such as a *cyclic redundancy check* (CRC) developed that consider the positions of the bytes

Near-Duplicate Detection … 1

- Harder task
  - Are web pages near-duplicates?
    With same text context but different
    advertising or formatting?
  - What about a page with a clock in a corner?
    Every crawl will be different!



Near-Duplicate Detection … 2

- Near-duplicate document defined

  using a *threshold value*
  for some *similarity measure*
  between pairs of documents

  – e.g., document *D1* is a near-duplicate of document *D2* if more than 90% of the words in the documents are the same

Near-Duplicate Detection … 3

- Two scenarios
  - Duplicates of a specific document D:
    - find near-duplicates of a specific document *D*
    - *O(N)* comparisons required
  - All pairs of duplicates:
    - find all pairs of near-duplicate documents in the collection
    - $O(N^2)$ comparisons

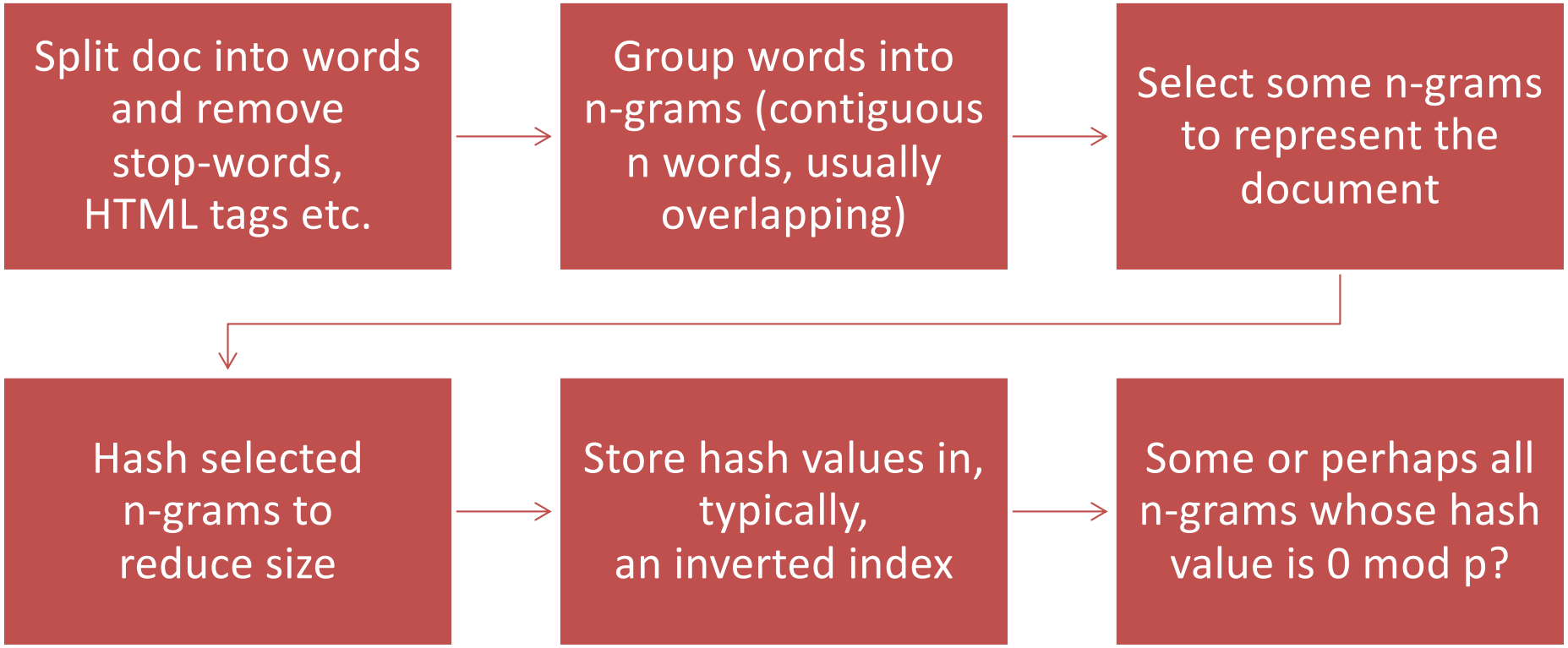
Near-Duplicate Detection … 4

IR techniques are effective for specific document scenario

For all pairs, other techniques used to generate compact representations of docs e.g. ‘fingerprints’

Fingerprints then compared

Creating a *Fingerprint* of a Document



Fingerprint Example
(n=3; 2 word overlap)

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

(a) Original text

tropical fish include, fish include fish, include fish found, fish found in, found in tropical, in tropical environments, tropical environments around, environments around the, around the world, the world including, world including both, including both freshwater, both freshwater and, freshwater and salt, and salt water, salt water species

(b) 3-grams

938 664 463 822 492 798 78 969 143 236 913 908 694 553 870 779

(c) Hash values

664 492 236 908

(d) Selected hash values using *0 mod 4*

Simhash

Word-based similarity comparisons more effective at finding near-duplicates

but not very efficient

Simhash combines the effectiveness of word-based similarity measures with the efficiency of hash-based fingerprints

Similarity of two pages (cosine correlation measure) proportional to the number of bits that are the same in their Simhash fingerprints

Simhash Algorithm

1. Process document into words and frequencies
   (terms and weights, more generally)
2. Assume we want a fingerprint of ‘b’ bits
3. Generate a b-bit (unique) hash value for each word
4. Initialize a b-dimensional vector
5. For each word W, update each component of V by

   Adding frequency of W to every component of V where the bit in the hash value of W is 1, and subtracting the frequency if it is 0
6. After all words processed, generate fingerprint by
   a. setting i^{th} bit to 1 if i^{th} component of V is positive, 0 otherwise

Simhash Example

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

(a) Original text

tropical 2 fish 2 include 1 found 1 environments 1 around 1 world 1
including 1 both 1 freshwater 1 salt 1 water 1 species 1

(b) Words with weights

tropical	01100001	fish	10101011	include	11100110
found	00011110	environments	00101101	around	10001011
world	00101010	including	11000000	both	10101110
freshwater	00111111	salt	10110101	water	00100101
species	11101110				

(c) 8 bit hash values

1 -5 9 -9 3 1 3 3

(d) Vector V formed by summing weights

1 0 1 0 1 1 1 1

(e) 8-bit fingerprint formed from V

REMOVING NOISE OR GETTING TO THE CORE CONTENT OF PAGES

Removing Noise

Many web pages contain ads, text, links, and pictures not directly related to the core content of the page

This *noise* could affect the ranking of the page

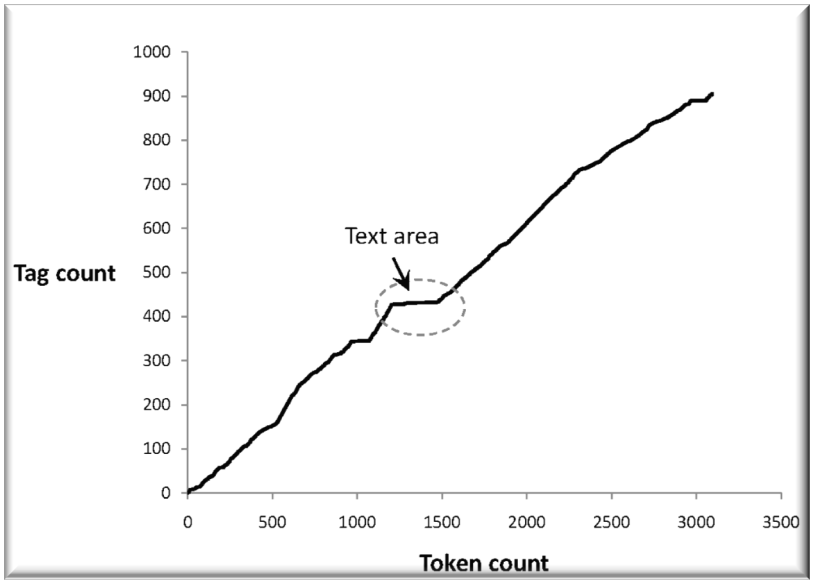
Techniques have been developed to detect blocks of core content on a web page

Non-core material is ignored or reduced in importance during indexing

Noise Example

Finding Content Blocks

- Fewer HTML tags in core content of page



Cumulative distribution of tags on sample page

- Main text content of the page corresponds to the "plateau" in the middle of the distribution

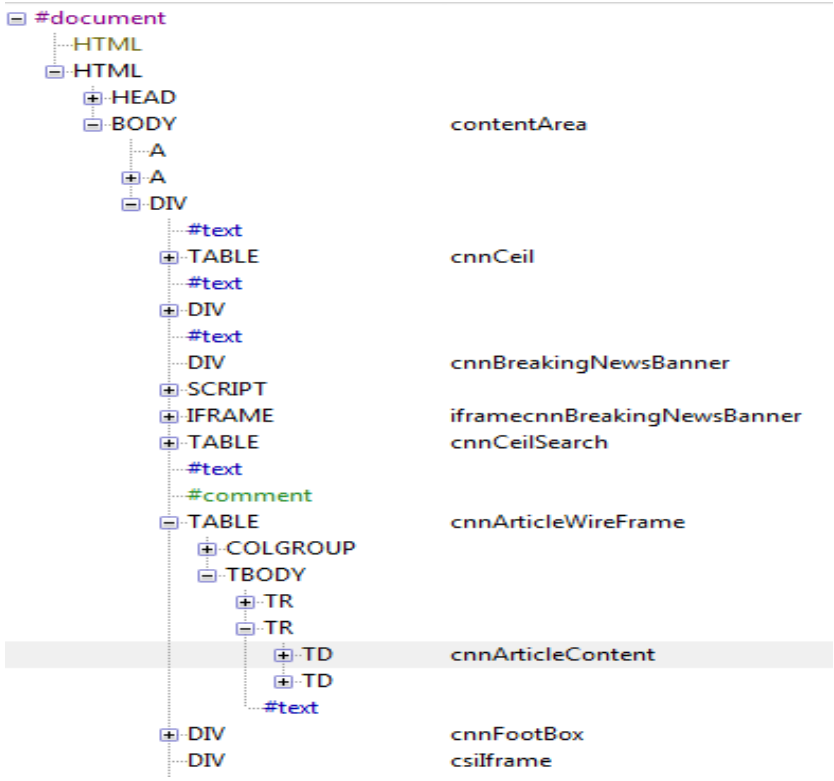
Finding Content Blocks ..1

- Represent a web page as a sequence of bits, where $b_n = 1$ indicates that the *n*th token is a tag
- Optimization problem where we find values of *i* and *j* to maximize both the number of tags below *i* and above *j* and the number of non-tag tokens between *i* and *j*
- i.e., maximize

$$\sum_{n=0}^{i-1} b_n + \sum_{n=i}^{j} (1 - b_n) + \sum_{n=j+1}^{N-1} b_n$$

Finding Content Blocks ..2

- Other approaches use DOM structure and visual (layout) features



Summary

- Covered Web Crawling, Crawling Metrics, Feeds vs. Crawls
- Looked at other types of Crawlers
- Discussed Document Conversion
- Detailed requirements for Document Storage
- Studied ways of dealing with Duplicates and Noise

- Next: **Transforming Data**

Readings

Croft, Metzler & Strohman (CMS) Chapter 3

Optional: Manning, Raghavan & Schütze (MRS)
Chapters 20 and 19
You can skip Sections 19.2.1, 19.2.2, 19.5, 19.6,
and Sections 20.2.2, 20.2.3, 20.4 for now

More Optional Reading:

Cho, J. and Garcia-Molina, H. (2003). Effective page refresh policies for web crawlers. ACM Transactions on Database Systems

http://dsg.uwaterloo.ca/seminars/notes/crawlingplus.pdf

ASSIGNMENT 1

CS 6200 Information Retrieval

Seattle/Fall 2019

Assignment 1 Web Crawling:
The Karen Sparck-Jones edition

Announced Sep 16, 2019
Due: Sep 30, 9am

Your task in this assignment is to implement a simple web crawler and apply it to crawl a portion of Wikipedia. The goal is to get you better aware of (some of) the work involved in crawling the web. In addition, once you have a crawler working, there's a lot of interesting projects you can do on the web.

Inputs and Outputs

1. The input parameters to the crawler will be
    a. a single seed URL **SeedUrl**,
    b. number of pages to be crawled **Numpages.**

2. The outputs to be submitted will be as follows:
    a. One or more **Python program files** which implement this assignment.
    b. A small (shell) script named **RunCrawler** (with an appropriate file extension like .sh , .py, etc.) which takes in the parameters SeedUrl and Numpages as inputs to create the desired outputs.
    c. A text file named **URLsCrawled.txt** which lists the URLs of the (up to) Numpages URLs that were crawled. This file should contain (exactly) one URL per line.
    d. A file named **stats.txt** with the maximum, minimum and average size of the files crawled, and the maximum depth (see below for definition of 'depth') you reached when you stopped. To keep it simple, use the following format:
    Maximum size: nnnnnnn bytes
    Minimum size: mmmm bytes
    Average size: aaaaaa bytes
    Maximum depth reach: d
    e. A file named **README.txt** which contains:
        i. A one or two sentence answer to the question: What was the most difficult part of this assignment?
        ii. Optional: Any additional information you may wish to provide about running the script **RunCrawler**.

What's expected to be done

3. Starting with the seed URL as depth 1, crawl each URL to a maximum depth of 5. That is, consider that the pages linked from the seed page are depth 2, the pages they linked to are depth 3 etc. Do not go beyond depth 5.
4. The crawler should stop when you have crawled **Numpages** unique pages (URLs) or if you are going beyond the maximum depth of 5.
5. **You may use standard libraries to get the contents of a document via HTTP, and follow HTTP redirects. (**So, it's OK to use urllib and re, but not BeautifulSoup or requests.)
6. However, *you must write your own code* to extract the links from each page, and to keep track of the pages you have crawled.
7. **You must also use politeness rules and wait at least one second between requests to the web server to get new pages.**
8. At all times, you must have a list (frontier) of URLs to be crawled next.
9. Because you know you will be crawling Wikipedia, you can optimize the crawling. Use the following rules to include or ignore links:
    a. Remember that web pages or web sites often use a base URLs, and further URLs within the web page/web site may start as relative URLs, say with "/wiki/…."
    b. *Include only links that start with* http://en.wikipedia.org/wiki/ This way you will only be considering Wikipedia articles in the English language, and will ignore other pages (which may or may not be HTML pages). **Note that the protocol may be http or https.**
    c. Ignore links to http://en.wikipedia.org/wiki/Main_Page . Most wiki pages have a reference to the main page, which we can ignore.
    d. Ignore links with a colon (':') after the host name. That is, ignore links like https://en.wikipedia.org/wiki/Help:Authority_control which are specific to Wikipedia.
10. To keep things simple, write code to retrieve pages one by one (in a single-threaded fashion). When you have retrieved a page, get its size (from the HTTP response, or by other means) and use it to calculate size statistics. Keep updating the file URLsCrawled.txt as you crawl each URL. At the end of crawling, compute and write out the stats.txt file.
11. Make sure the code is adequately commented.
12. When the coding is completed, run the program with the following values of parameters:
    SeedUrl = https://en.wikipedia.org/wiki/Karen_Sp%C3%A4rck_Jones
    and Numpages = 1000
13. One way to do this: as you crawl the URLs, save all the files crawled in a folder, with each file having the filename nnn.txt where nnn is a number from 1 to Numpages.
14. Be sure to follow the instructions above. You may lose points if you miss any of these instructions. Also, keep in mind the academic integrity rules we discussed.

Important note: We will be building upon this code and using the output of this code in the next assignment(s). So, retain the code, and save the crawled documents as well.

Assignment Notes

Usage of Programming Libraries

- Urllib, re OK
- BeautifulSoup or requests not OK for this assignment

How will you check for duplicates?

- URL vs. content

Assignment submission details

Questions?